Adding Value to Travel Behavior Surveys: The Network Analyst Approach

METROPOLITAN
TRANSPORTATION
COMMISSION

Adding Value to Travel Behavior Surveys: The Network Analyst Approach

ESRI Paper UC1577

August 2006

Contributors:

Agency staff members from the Association of Bay Area Governments and the Metropolitan Transportation Commission contributed to the research and materials contained in this paper.

Authors:
Ms. Stella Wotherspoon
swothe@mtc.ca.gov

Mr. Kearey L. Smith
ksmith@mtc.ca.gov

Metropolitan Transportation Commission
101 8th Street
Oakland, CA 94607
Telephone: 510.817.5700

SAN FRANCISCO BAY AREA REGION

Regional Statistics

Total Area:	7,100 Square Miles
Total Population:	6,783,762
Jurisdictions:	9 Counties, 101 Cities
Largest Cities:	San Jose, San Francisco, Oakland
Transportation System:	620 miles of Freeways, 518 Miles of Railways. 22 Transit System Operators 24.760 Transit Stops

Table of Contents

Adding Value to Travel Behavior Surveys:
The Network Analyst Approach

Abstract

This research paper describes the GIS tools and methods used to examine travel patterns as reported in the Bay Area Travel Survey conducted in 2000 (BATS2000). The BATS2000 provides a comprehensive picture of regional and sub-regional travel characteristics for over 15,000 households, and is the only database that allows analysts to examine the full picture of both work and non-work travel patterns within the nine-county Bay Area region. The BATS2000 survey contains detailed information on the precise origins and destinations of all trips, including precise home locations for survey respondents. This level of detail is used to create travel behavior models based upon point-to-point travel times and distances, as opposed to zone-to-zone. This research paper will discuss the array of Network Analyst tools used to conduct analyses such as door-to-door distance measurements, walk/drive distance to nearest transit stop, household-level neighborhood density & accessibility measurements, among others.

1. Introduction

In 2005, the Metropolitan Transportation Commission (MTC) conducted a series of GIS analyses of the Bay Area Travel Survey 2000 (BATS2000), a large-scale regional household travel survey of the nine-county San Francisco Bay Area region in California. The goal of these analyses was to augment the BATS2000 dataset with geographic variables derived from GIS analyses using software developed by Environmental Systems Research Institute.

BATS2000 was a $1.5 million study to obtain detailed travel and activity statistics from 15,064 Bay Area households. BATS2000 was the fourth in a series of major travel surveys conducted in the Bay Area over the past forty years, starting with the original 1965 Bay Area Travel Survey; and continuing with other major

surveys in 1981, 1990 and 1996. The BATS2000 dataset is a centerpiece for new sets of travel behavior models that will be produced over the next several years. The survey also provides a comprehensive picture of regional and sub-regional travel characteristics and is the only database that allows analysts to examine the full picture of both work and non-work travel patterns within the region.[1]

In contrast with previous Bay Area household travel surveys, BATS2000 was an activity-based survey where respondents recorded all activities, including trips, over a two-day period. Respondents provided origin and destination addresses or nearest intersections for each trip which were later geo-coded, yielding approximately 229,000 intra-regional origin-destination pairs, of which approximately 198,000 were used in this study. Because BATS2000 contains detailed information on the precise home locations, and origins and destinations of all trips, it is a significant improvement over the prior MTC BATS surveys. The prior surveys only retained census tract and travel analysis zone (TAZ) information on trip origins and destinations. This is a very important upgrade: new sets of MTC travel behavior models can now be based on precise point-to-point travel times and distances, instead of more generalized zone-to-zone measures.

Past analyses measured trip length using a straight line or "as-the-crow-flies" method. The ESRI ArcGIS Desktop extension, Network Analyst 9.1, enables an alternative calculation method that utilizes the street network and traffic rules such as one-way restrictions. This is a second important upgrade as the point-to-point travel times and distances generated by these analyses represent an approximation of the respondents' likely path of travel, instead of the generalized straight line and zone-based calculations of past analyses.

---

[1] The final survey consultant report for the BATS2000, and early releases of the data files, are available on MTC's web page at: http://www.mtc.ca.gov/datamart/survey.htm.

2. Summary of Analyses

The most significant task accomplished by this study was the calculation of the door-to-door trip network path, mileage, and time for both walk trips and drive trips. In order to derive this important geographic variable, research was conducted into various ESRI based analytical tools and methods that could be used to perform this analysis. ArcView 3.2 Network Analyst contains some tools that could be used to analyze individual or small groupings of origin and destination trip pairs. However, in order to solve the approximately 198,000 intra-regional origin and destination pairs contained in the BATS 2000 database, it was necessary to explore automation routines that could be used to solve the trip paths in a batch mode. When Network Analyst 9.1 was released, it included several modeling and scripting capabilities that allowed for the development of automation tools that could be used to solve multiple trip paths.

Other important tasks accomplished by this study include the geocoding of all origin and destination trip pairs, deriving the walking distance and time to the nearest transit stop, calculating the number of households and work locations within one-quarter, one-half, or one mile of a transit stop, as well as the calculation of residential and employment densities within a half-mile network buffer of all households, work locations, and transit stops used in the study.

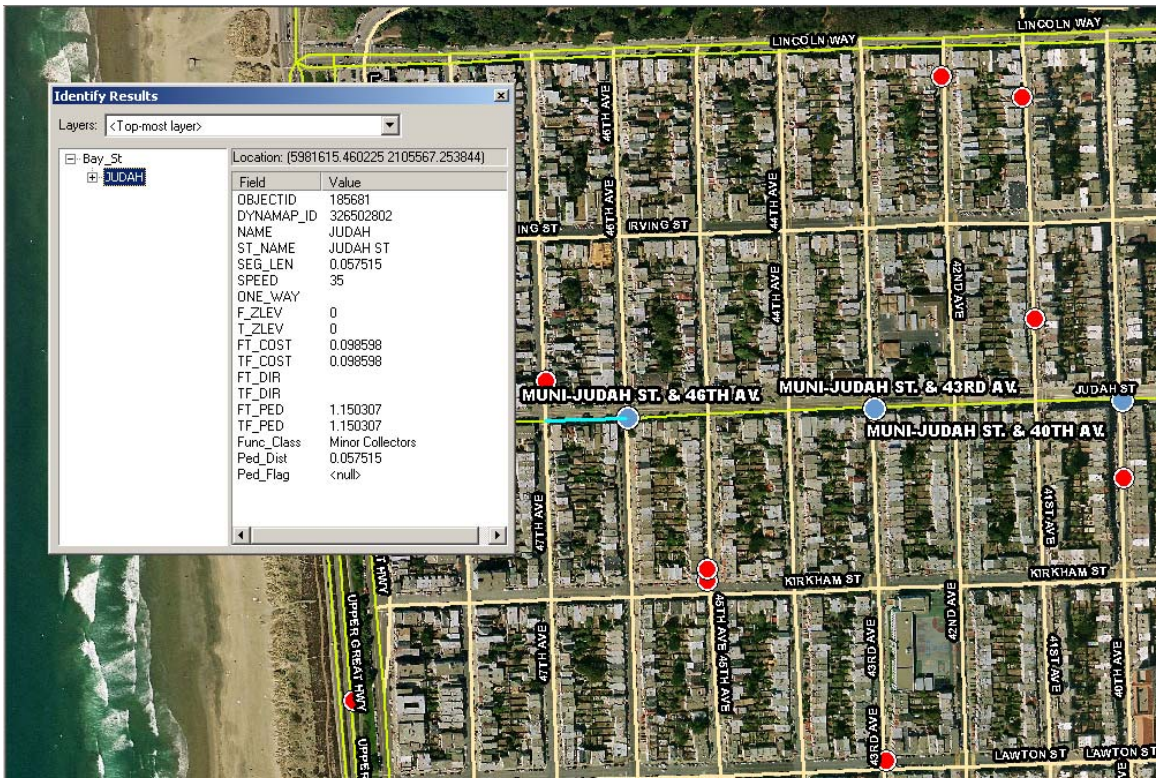
While the majority of the work was performed using the Network Analyst 9.1 extension, several desktop ArcGIS functions were used such as spatial joins and geocoding of intersection and address data collected in the study. Network Analyst 9.1 contains several functions, or solvers, that were used to do the following types of analyses:

1. Build Network Datasets, and Calculate Network Locations for all study points used in the analysis (Origins and destinations, transit stops etc.)
2. Find the Best Route for select trips collected in the BATS 2000 survey
3. Calculate the nearest stop or location using the Origin and Destination Cost Matrix tools
4. Create network buffers (walking buffers as opposed to circular buffers) of all network locations used in the study

However, some of these solvers can only process one set of records at a time. For example, the Find Best Route solver can be used to generate the door-to-door trip network path, mileage, and time for a single origin-destination pair at a time. To process the total number of individual trips (198,477) using the ArcGIS user interface would be a very time-intensive and manual process. Batch processing can be handled, however, using custom Python or VBA scripts that utilize either the Geoprocessor or ArcObjects object libraries.

The analyses were focused on pedestrian and vehicular network paths and buffers. The source data used was Teleatlas NA-GDT Dynamap/Transportation, a vector-based street network database in which streets are represented as line segments. Each street record contains attributes that describe characteristics of the street, such as address range, one-way restriction, average speed, length (miles), and cost (the travel time assuming average speed) and classification. See Figure 1.

Figure 1. Sample Street Record with Values



The majority of street records in this database are accessible to motorized vehicles. Of these streets, there are several types where pedestrians are not permitted, such as highways and highway ramps, but there is no encoding to indicate this condition. The pedestrian analyses necessitated the creation of new attributes for the street records, pedestrian time and pedestrian distance, to model this use of the streets, as well as the development of a network dataset with impedances for both vehicular and pedestrian modes of travel. The street network database was also augmented with pedestrian-only paths in the vicinity of transit stations.

3. Data Collection and Pre-processing

Building a Street Network

Problem Statement

Develop a dataset of the streets of the nine-county Bay Area region that supports modeling of vehicular and pedestrian networks, the generation of least cost routes, the creation of network-based buffer zones, and the identification of closest facilities (transit stops).

Solution Method

MTC analyzed street network databases from several commercial providers and ultimately licensed the Teleatlas NA-GDT Dynamap/Transportation database. The Dynamap/Transportation database format is ArcView shapefile. The nine-county Bay Area region dataset was assembled from Dynamap/Transportation county shapefiles provided in a geographic projection, North American 1983 datum, and projected to NAD1983-UTM-Zone 10N, meters. This resulted in over 740,000 polyline features in the database, representing roadway segments for the entire region.

Modifications were made to the database to support modeling of pedestrian usage. These modifications included adding street records for pedestrian-only

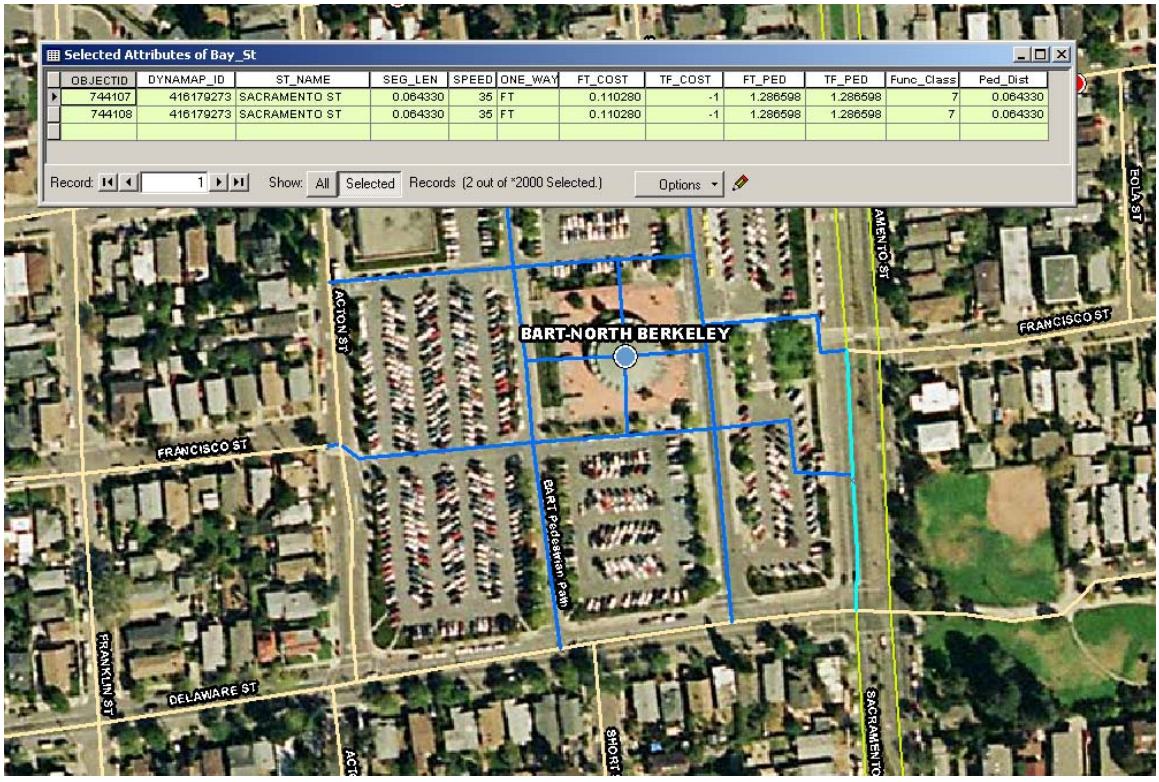
paths around transit stops and adding attributes to existing street records that represent pedestrian distance and time values.

Pedestrian-only paths were primarily added in the vicinity of Bay Area Rapid Transit (BART) stations. This was necessary because some of these stations are surrounded by large parking lots. The original street database did not contain road segments that indicate where pedestrian access exists in these lots. Therefore, it was necessary to add network segments to the database in order to model pedestrian access for these areas. Pedestrian paths were also added to connect the street network to transit stops that were not coincident with the street. This was necessary for several BART stations where the station entrance is in the center of a parking lot and lot access roads were not modeled in the database. These new geographic features were digitized based on AirPhoto USA aerial imagery as well as field observations. See Figure 2.

Figure 2. North Berkeley BART Station Showing Added Pedestrian Paths

Initially, after building the network dataset, we experienced connectivity problems with the new segments. We discovered that there were digitizing errors that created invalid line segment topology. In order to correct this problem, we used vertex snapping at intersections. In addition, we discovered that existing street segments that intersect new street segments had to be split at the existing segment's vertex. Once all the segments were added, and topology was verified, it was necessary to add the appropriate attribute values that represent the network impedances used in the analyses. See Figure 3. The area highlighted in light blue represents the street segment where the attributes were recalculated.

Figure 3. Street Features to Split and Recalculate Attributes



Because the original database is oriented toward the use of motorized vehicles, and did not contain sidewalks or other representations of pedestrian right of way along roadway segments, it was necessary to add attributes that represent the cost of travel to pedestrians using the roadway features. The three attributes PED_DIST (pedestrian distance in miles), FT_PED (pedestrian cost in minutes in the from-to direction), and TF_PED (cost in the to-from direction) were added to the database.

PED_DIST was set to the same value as that for motorized vehicles, except for roads where pedestrian access was restricted. These segments received a constant value of –1. Network Analyst 9.1 interprets street records with negative integer values for distance or cost as inaccessible to travel and will not include these segments in routes. FT_PED and TF_PED represent the cost in minutes for a pedestrian walking 3 miles/hour along the street segment. In cases where pedestrian access was restricted, a constant value of –1 was entered for these attributes. The values for FT_PED and TF_PED for a street record are identical, indicating that the same cost is levied on a pedestrian traveling in either direction, or put another way, there is no concept of asymmetrical or one-way pedestrian access to a street segment.

Initially we used the speed attribute to identify the street records that are restricted to pedestrians using the criteria of speed greater than 35 miles/hour. However, this was not sufficient to model the actual accessibility of Bay Area roads. We effectively removed highways, but retained highway off-ramps where the speed was 20 m.p.h. as well as unnecessarily restricted pedestrian access to some high-speed primary roads.

A more accurate method of modeling pedestrian access was established after review of three classification attributes in the Dynamap/Transportation database: ACC or Arterial Classification Code, FCC or Feature Class Code, and speed. ACC is Teleatlas NA/GDT's system of road categorization that reflects the level of mobility a road provides within the network. Mobility measures are derived from the volume of traffic on a road segment and the length of trip it serves. A low ACC value indicates a road with high level of mobility. ACC can be used to establish routing hierarchies. FCC is Teleatlas NA/GDT's method for detailed categorization of types of road features by characteristics such as separated and unseparated.[2] To conform the categorization of roads to a standard used within MTC, a new

[2] Tele Atlas NA/GDT, User Manual--Dynamap/Transportation v. 15 (Boston: TeleAtlas, 2005).

attribute, Functional Class, was added and a concordance table mapping ACC, FCC, and speed was developed. See Table 1.

Table 1. Concordance Table for Functional Class

ACC	FCC	SPEED	Code	Func_Class	Description
1	A15	65	1A1565	1	Major Arterials
1	A63	20	1A6320	2	Major Arterial Ramps
2	A10	55	2A1055	3	Minor Arterials
2	A60	20	2A6020	4	Minor Arterial Ramps
3	A11	55	3A1155	5	Major Collectors
3	A60	20	3A6020	6	Major Collector Ramps
4	A21	35	4A2135	7	Minor Collectors
4	A60	20	4A6020	8	Minor Collector Ramps
5	A30	35	5A3035	9	Local Roads
5	A60	20	5A6020	10	Local Road Ramps
5	A50	1	5A501	11	Other Thoroughfare
5	A51	1	5A511	12	Pedestrian Paths

The FCC values A60 and A63 indicate access ramps, the former not associated with a limited access highway and the latter a cloverleaf or limited access interchange. Street records with these FCC values were assigned to specific Functional Class levels so they could be distinguished from non-ramp segments.

The PED_DIST, FT_PED and TF_PED values for street records with values of Functional Class of 4 or less were set to –1, indicating their inaccessibility to pedestrians.

The Teleatlas NA-GDT Dynamap/Transportation shapefile dataset includes duplicate features that are used to store alternate names for certain street records such as San Pablo Avenue, a.k.a. California State Route 123. These line features are coincident, and are indicated by a value of –9 in the elevation attributes, F_ZLEV and T_ZLEV. We deleted these street records from the database to eliminate the incorporation of these erroneous line segments into the network dataset.[3]

Network Dataset

Problem Statement

Create a network dataset that Network Analyst 9.1 can use to perform network calculations.

Solution Method

A network dataset is the file used by Network Analyst 9.1 to calculate routes and service areas, to locate closest facilities, and to produce origin-destination matrices. A network dataset can be created from line features, point features and turn tables that model transportation networks. Typical components of these features used in a network dataset include: streets or paths represented by polylines, attributes that can be used to define path connectivity policies, street or path names, elevation of path segments, functional hierarchy of streets, as well as some form of network impedance such as segment length or estimated speed of travel. The process of building a network dataset produces network attributes, which are used to model impedances, restrictions, and hierarchy for the network.

[3] ESRI White Paper, Preparing Street Data for Use with the Network Dataset (Redlands: ESRI, 2005).

Subtypes of street features were categorized in the Dynamap/Transportation database using the new attribute, Functional Class. These subtypes are used to establish the network dataset connectivity policy. As all street features in the database have endpoint vertices at junctions with other street features, the connectivity policy was defined as 'End Point'.

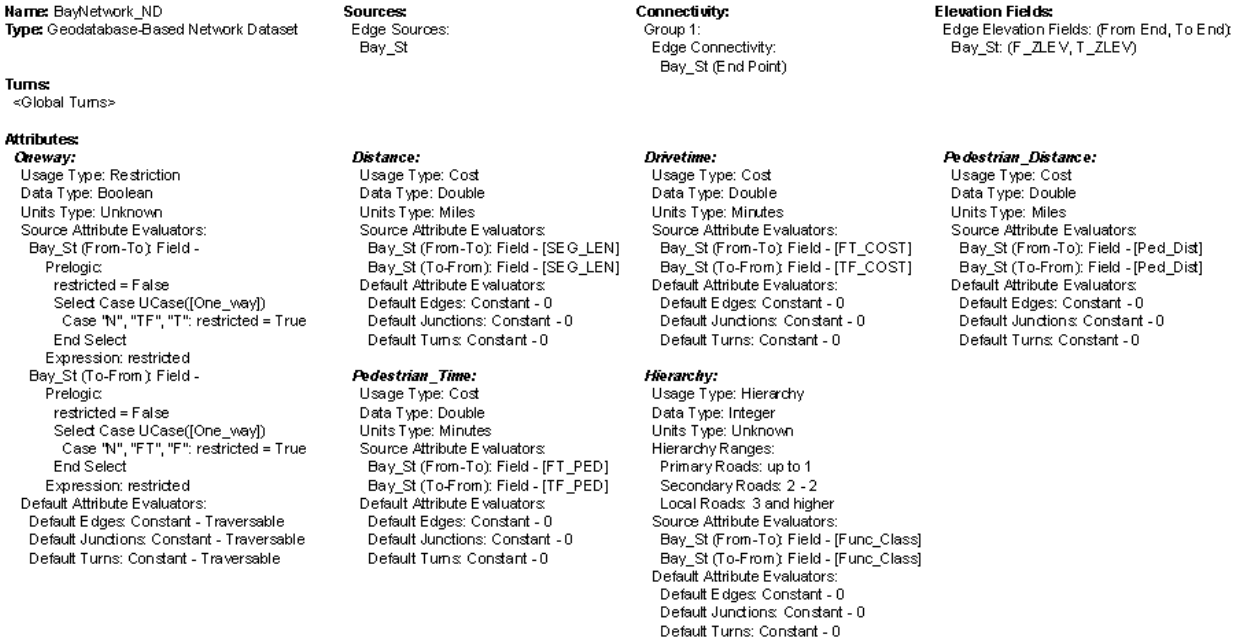
The Dynamap/Transportation database includes attributes that define the elevation of the endpoints of each street record. These attributes, F_ZLEV and T_ZLEV are used to model connectivity. For two street records to route correctly the elevation values for the connecting records must be the same. An overpass has a different elevation value from the street that runs beneath. In this case, the differing elevation values would not allow connectivity between these roads.

The Dynamap/Transportation database includes a turn table that gives detailed information about restricted maneuvers, such as time-restricted left turns, no u-turns, etc. This table was not used as a source in building the network dataset. However, the dataset was prepared for future use of the turn table by modeling turns using the "Global Turns" turn source.

Network attributes were created for impedances that were used in the analyses we conducted, as well as for other properties of network source elements such as hierarchy and one-way restrictions. We defined network attributes for distance, drivetime, pedestrian distance, and pedestrian time, hierarchy, and one-way restrictions. Each network attribute was mapped to an evaluator attribute in the street database that provides values used in calculating distance or time based on the network attribute or impedance selected: Distance was mapped to SEG_LEN, Drivetime was mapped to FT_COST and TF_COST, Pedestrian Distance was mapped to PED_DIST, Pedestrian Time was mapped to FT_PED and TF_PED, and Hierarchy was mapped to Functional Class.

The network dataset was not configured to generate driving directions, as the analyses did not require these data. A summary of the network dataset settings described in this section follows in Figure 4.

Figure 4. Summary of Network Dataset Settings



Geocode Households and Trip Origin Destinations

Problem Statement

Obtain x-y coordinates for BATS2000 respondent household addresses as well as for origin and destinations of each trip.

Solution Method

BATS2000 is an activity-based survey where respondents recorded all activities, including trips, over a two-day period. Respondents provided origin and destination addresses or nearest intersections for each trip which were geo-coded, yielding approximately 229,000 intra-regional origin-destination pairs. Morpace International, a survey research firm contracted to administer the survey, performed the geocoding of the households and trip origin-destination pairs. The geocoding success rate for the households was close to 100%. The trip origin-

destinations presented a greater geocoding challenge as many were reported as intersections. The success rate of the trip origin-destination pairs was 93.4%.[4]

On receipt of the data, MTC staff performed geocoding using a TIGER basemap so census geographies for the locations could be accurately appended to records. This geocoding was repeated using a TANA/GDT street basemap that MTC adopted in January 2003. This included detailed x-y coordinates, in NAD83-UTM-Zone 10N, meters; as well as standard Census Bureau and MTC geography including: block, block group, census tract, public use microdata area (PUMA), MTC 34 superdistrict, MTC 1454 travel analysis zone, and county. (Note the census geography has been adjusted by GDT to align with the GDT street layers.) This geocoding effort was a further improvement over previous ones as an offset was used that placed the points on either side of the street centerline and not on the boundaries of any census geographies.

Obtain Transit Stop Features

Problem Statement

Obtain transit stop point features from Bay Area transit operators.

Solution Method

The transit stop point features used in this analysis are the result of an MTC traveler information initiative called the Regional Transit Information System (RTIS). The RTIS includes a number of projects designed to provide up-to-date transit information to the public and to MTC's transportation partners. At the heart of the RTIS is the Regional Transit Database (RTD), a spatially enabled relational database containing current transit stop and route features and schedule information for over 40 transit operators in the region and supports the TakeTransit trip planner (http://transit.511.org/tripplanner/index.asp). Route,

[4] MORPACE International, Inc., *Metropolitan Transportation Commission, Bay Area Travel Survey 2000 Final Report*, (Farmington Hills: MOREPACE, 2002).

stop, and schedule data are provided to MTC by transit providers several times a year. Over 24,000 transit stop features were converted to the ArcGIS shapefile format. A subset of 406 passenger rail and ferry stop features was primarily used in these analyses.

4. Network Analyses

Batch Processing of Door-to-Door Trip Polylines

Problem Statement

Generate network-based optimized routes for every trip record in BATS2000 using the impedance of distance. Create polylines of these paths and record the accumulated time and distance for each route. Door-to-door trip distances and elapsed time are a valuable addition to the BATS2000 dataset. The survey respondents reported trip travel times that could be compared against the estimated times for the routes generated by this study. Additional statistics on average trip length and trip length frequencies can be derived from the trip distances. The aggregate vehicle miles of travel can be checked against reported odometer readings. Last, the values can be used for estimating the non-motorized and motorized components of future choice models.

Solution Method

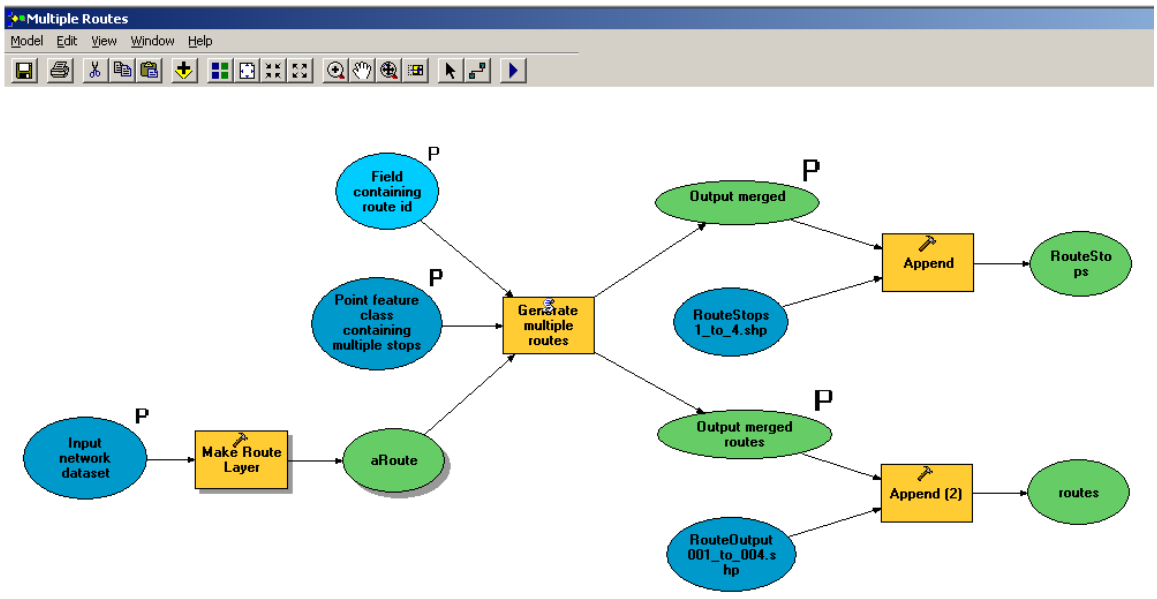
Network Analyst 9.1 contains a solver that will calculate the best route between two or more points, depending on the impedance chosen. The BATS2000 source data for the trips stores origin and destination coordinates as attributes of a single trip record. Because Network Analyst 9.1 requires stops to be loaded as individual records, preprocessing was performed in PC-SAS to reassemble the source trip records to a consecutive stops file with every origin and destination represented as a distinct record. However, loading 396,954 stops, which represent 198,477 intra-regional weekday trips, into a single route yields extraneous polylines. For example, this approach will create a polyline from the destination of trip "A" to the origin of trip "B" that is meaningless to the analysis. Running this many

consecutive stops is also a large processing task. The best approach to this problem is to load the origin and destination for a trip, solve the route, save the polyline and accumulated attribute values to a feature class, and then repeat the process with the next trip until there are no more trips.

An iterative process for solving all 198,477 trips was implemented using a Python script and the Geoprocessor object model and, later, a VBA script using the ESRI ArcObjects object model. Both of these scripts were provided to MTC by the ESRI Network Analyst Development Team staff.

ArcGIS 9.1 contains a spatial model development tool called ModelBuilder. ModelBuilder allows users to develop simple to very sophisticated models using the geoprocessing tools contained in the ArcGIS development environment. ModelBuilder can be used to document as well as automate the workflow used to solve a problem using GIS. ModelBuilder in ArcGIS 9.1 can be used to automate a sequence of processes, but a script must be used to perform iterations of the processes. See Figure 5.

Figure 5. Door-to-door Trip Polyline Model

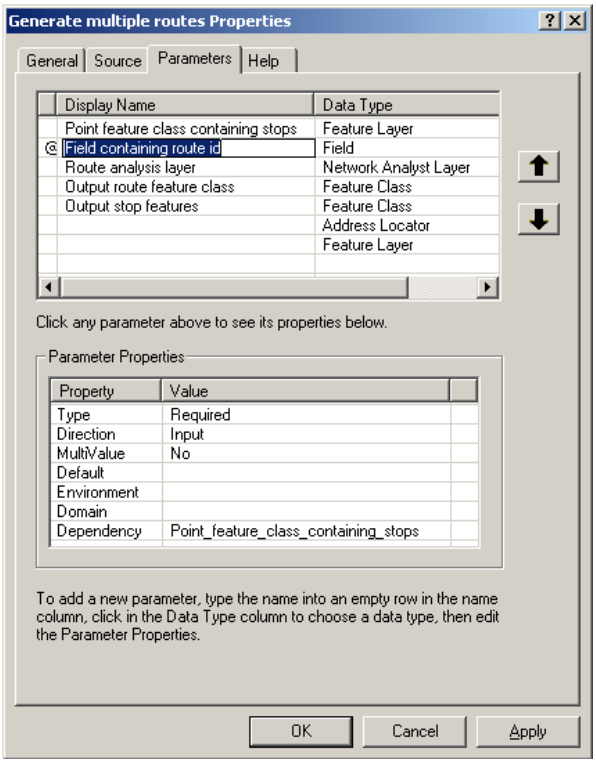


In this model, the network dataset, BayNet_ND, and a point feature class, OD_HH, are inputs. OD_HH contains the origin and destination points as x-y coordinates. A

route analysis layer is created from the network dataset, which is used by the Python script, Generate Multiple Routes. See Appendix A. The script performs the route analysis and writes the route polylines and stops to a shapefile. Because the script was run on batches of origin destination pairs, the last step in the model appends the two script output files to master files.

The script expects five system arguments: the point feature class containing the stops for the trips, the field containing the trip id for each stop, the network analysis layer, the path for the output stops file, and the path for the output routes file. These are set up as properties of the script tool. See Figure 6.

Figure 6. Script Tool Properties

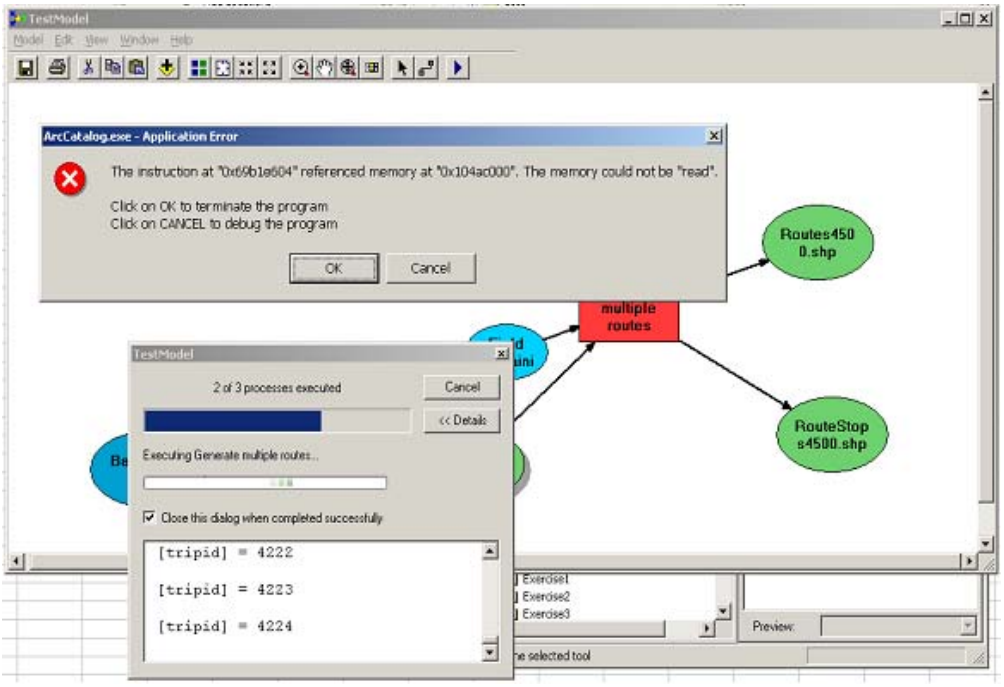


The script contains several defined functions that are called from within the main module, e.g., **getTempRoute()**. As the script runs, first the field object for the trip id field is obtained and the field type is translated. Then the script creates a temporary table to hold the results of the Frequency tool. This tool is used to scan the OD_HH file and create a list of unique trip id number values as well as the frequency with which the values occur. The OD_HH file contains one row for each origin and destination and each row includes an attribute that holds the trip id number. Therefore, the result of the Frequency tool for each trip id number value is two. The trip id field is added to the analysis layer so each route is identified with a trip id. A temporary scratch shapefile is created to store the route outputs. The routes and stops sublayers are named so the features they contain can be copied. A search cursor is opened for the temporary table containing the frequency data. A

loop is initiated for each unique trip id. Within this loop, the script creates a table view that contains the origin destination records for the current trip id. These stops are added to the to the analysis layer and the route is solved. The route is copied to the temporary shapefile and the trip id field is added. The temporary features are then appended to the output shapefiles and the script loops to the next trip id in the search cursor. Once the script processes all the trip ids in the search cursor the temp files are deleted and the script terminates.

The Generate Multiple Routes script ran within ArcCatalog 9.1 with inconsistent performance. The average processing time of each route was 1.5 minutes, however ArcCatalog would throw unhandled exceptions. See Figure 7. There was no obvious cause as the outputs varied in size from 2 records to over 1,000 records. We could not take advantage of overnight or weekend processing because there was little possibility that the script would run uninterrupted.

Figure 7. Unhandled Exception



Attempts to work around this problem included creating a feature layer input with a subset of stops, creating a shapefile network dataset, and running the process on a different PC. None of these strategies resolved the problem of unhandled exceptions. A suggestion by ESRI staff to run the script using an ArcSDE

network dataset was not tested due to the version of SDE running on the MTC server. Network Datasets are only supported at the ArcGIS 9.1 (or later) level. Later review of the code by ESRI staff indicated that the script failure point occurred when setting the shapefile dbf string field value, or when the route was appended to the output file. It was not determined why this error occurred inconsistently.

A side effect of the unhandled exception issue was that the temporary table and shapefiles were not deleted. The code controlling the deletion of the temp files appears in the script outside the while loop. Because the script failed within the loop, the temp folder was not cleared of these files. If these files were not manually deleted, subsequent runs would fail when the script tried to create new versions of the files.

After troubleshooting these issues, the script processed one trip and then failed with a message that the table view already existed from the previous iteration. It is possible that this bug was introduced to the script inadvertently after modifications, but the cause for this event is unknown. The Make Table View tool is supposed to overwrite an existing table view if the same table view name is entered, however this was not occurring. The work around for this was to insert code for deleting the table view prior to the creation of the next table view.

Periodically, after multiple script runs, the script would fail to run due to the network dataset reverting to a status of "Not Built". The cause of this problem was tracked to the personal geodatabase that contained the network dataset. The geodatabase size grew with each successive run of the script and eventually the size exceeded the 2.1 GB maximum allowed by Microsoft Access. The reason for this growth was that MS Access was not reclaiming space from temporary tables created with each iteration. This issue was resolved by using the Compact Database tool. The script was amended to perform this sub-process every 100 routes. This additional sub-process extended the average processing time for each route to over 2 minutes. The time required to complete processing the trip dataset at this rate was 6,615 hours or 275 twenty-four hour days. Thus, it was necessary to improve the performance of the process.

ESRI Network Analyst Development Team staff reviewed the datasets and Python script and proposed using a VBA script and the ArcObjects library to improve performance. The reason the Python script had poor performance was still undetermined but moving away from use of the Network Analyst Geoprocessor object model was an alternative approach.

The VBA script, MultiRoute, uses a different approach than the Python script to solve the routes. See Appendix B. This script performs one sequential scan through the origin destination file, performs the route analysis for each pair, and uses an insert cursor to write the route polyline and accumulation attributes to an output shapefile before continuing with the next pair. The script terminates when no more paired origin destination records are found in the input table. Temp tables are not used in this process.

This script is added as a macro to the Visual Basic Editor in ArcCatalog. The ESRI Network Analyst Object Library must be selected as an available reference. The script expects the stops feature class to be at the root level of the personal geodatabase that contains the network dataset. The network dataset must exist within a feature dataset. To launch the script, the stops feature class is selected in ArcCatalog and the script is run (Tools > Macros > Macros > Run). The status bar shows a trip id counter for monitoring the progress.

The VBA script ran an order of magnitude faster than the Python script (10 routes per minute), which cut the processing time requirement down to 13 twenty-four hour days, a significant improvement. However, there were still some bugs to resolve. The script caused ArcCatalog to hang when errors were encountered that the error handling code could not manage. Modifications to the error handling code resolved these issues. In the first iteration of the script, the stops feature class was not sorted prior to the table scan. This resulted in script failures because, in several places in the stops feature class, the physical order of the origin destination records was not sequential. The script would stop when a matched origin-destination pair was not found. The addition of a table sort by ascending trip id at an early point in the script resolved this issue.

To further speed the processing of the trips, the network dataset hierarchy was used. The hierarchy attribute evaluator was set to the Functional Class attribute and the network was rebuilt. When hierarchy is used for long trips, the route solver is more efficient, as fewer network elements are examined. After implementing the hierarchy the script processed a majority of the 198,477 records in about 47 hours (60 routes per minute). The error log captured reasons some trips were not solved; the primary reason was that stops were located on non-traversable elements. These stops were manually relocated and the routes solved successfully.

An alternative solution to the issue of stops located on non-traversable elements is to amend the script to add locations using previously calculated values and to exclude the non-traversable sources in the location calculation process. ESRI staff provided some sample VBA code for this process that needs to be modified so it will run within the MultiRoute script.

After processing the trips, the routes were checked for accuracy. This was done by examining the paths for a sample of trips and by comparing the calculated route distances with respondent-reported distances. Manual examination of the roadway segments that made up several routes did not turn up any erroneous routes. However, the comparison with respondent-reported trip distances revealed 15,047 trips where the difference between the calculated route distances and the reported trip distances was greater than five miles. The worst cases were examined and an error in the hierarchy setup where some major arterial ramps were incorrectly coded was identified. These street records were corrected, the network was rebuilt, and the entire dataset was run through the script again and quality checked.

Buffer Analyses of Household and Work Location Proximity to Transit Stops

Problem Statement

Determine if respondent's home and work locations are within a quarter, half, or one mile walking distance of passenger rail and ferry transit stops and append smallest distance and stop name to household and work location feature classes. For example, if a household is within a quarter mile of a station it is also within a half or one mile from the station but the household record will contain only the quarter mile designation.

Solution Method

Network Analyst 9.1's Service Area solver was used to create walking distance network buffers around each transit stop. The Service Area function uses the network dataset and a point feature class as inputs. The point features represent facilities, or transit stops, for which service areas will be calculated.

The network locations for the transit stops were calculated in advance and these calculated locations were added to the service area analysis layer. A definition query was used to present only stops for a single transit agency, e.g., BART or MUNI. The service area buffers were generated by agency using an impedance of pedestrian distance and default breaks of one-quarter, one-half and one mile. The direction of travel was toward the facility, U-turns were permitted, and one-way driving restrictions were not followed. Overlapping polygons were not merged and rings were generated. The polygons were generalized. The resulting buffers were saved as three layer files, each representing one of the break intervals.

The buffers were added to ArcMap and placed in a group layer by transit agency. The households point feature class was added to the project and spatial joins were used to identify which buffers the household point features fell inside. The spatial join was of the type polygons to points and resulted in each household point feature receiving the attributes of the buffer that it falls within. The output files were saved and merged to create a master household feature class containing the

attributes of the smallest buffer a point fell within. This process was repeated for the work locations feature class.

Note that when transit stops are closely spaced, the resulting service area polygons may overlap. The spatial join process described above joins a point with only one of the overlapping polygons with which it is coincident. This result was acceptable for the further analyses to be conducted with these data, therefore, no measures were taken for special handling of the overlapping polygon joins. For more information about how processes with overlapping polygons can be handled, see the section Calculation of Population and Employment Counts within One-half Mile Buffer around Transit Stops and Households.

Generalized polygons were used in this analysis because less processing time is required and service areas were created for a large number of facilities. Generalized polygons have more linear boundaries due to extrapolation between a small set of sample points representing the distance from the facility and may slightly exaggerate the accessibility around a transit station. Detailed polygons can have more elaborate boundaries as a greater number of sample points are generated.

Analyses of Distances from Households to Nearest Transit Stop

Problem Statement

Determine the walking accessibility of households and work locations to the closest bus or rail transit stop within one mile. These data will be used to more accurately predict the effect of proximity to transit on an individual's choice of travel mode and to further refine travel model estimates.

Solution Method

The Origin Destination (OD) Matrix solver was used to calculate the network walking distance from each household to the closest transit stop. The network locations of the household were calculated using the Calculate Locations tool. These locations were then added as origins and the transit stops were added as

destinations to the OD analysis layer. The impedance was set to pedestrian distance. The default cutoff value was set to one mile, which allows the solver to ignore destinations beyond that distance from an origin. The destinations to find parameter was set to one so only the closest destination within one mile was found. U-turn and one way driving restrictions were not enforced. A hierarchy was not used.

This analysis was run with 15,064 households as origins and 24,760 transit stops as destinations on a PC with 1 GB RAM and never successfully completed. An OD matrix problem of this magnitude run against a large network dataset requires a large amount of RAM. If RAM is limited, the application is likely to run out of memory. [5] A workaround was to split the problem into smaller ones by using a 15,000 meter grid to partition the region's extent. The OD Matrix solver was run on the households and transit stops within one cell and then the outputs were appended into one file. To overcome boundary issues, or a closest destination that was outside a cell partition, a select by location was first performed on the destinations within one mile of the cell. This superset of destinations was processed with the household records that were located within the cell.

An alternative approach was to run a subset of households against the transit stops and complete the process in batches. The largest OD matrix problem we were able to solve was 250 X 24,760 so a script was required to automate the process. The manual processing of these data using the grid approach was minimally laborious, so a script was not developed.

An OD Matrix was also produced for the 15,064 households and the closest of the 406 passenger rail and ferry transit stops. This was a manageable number of inputs for the solver and was processed successfully in one attempt.

---

[5] Hierarchical Routes in ArcGIS Network Analyst, ESRI White Paper, 2005." with "ESRI White Paper, *Hierarchical Routes in ArcGIS Network Analyst* (Redlands: ESRI, 2005).

Calculation of Population and Employment Counts within One-half Mile Buffer around Transit Stops and Households

Problem Statement

Using population and employment data by census tract from the Association of Bay Area Governments demographic forecast, *Projections 2005*, estimate the total population and jobs in 2000 within a half-mile pedestrian network buffer surrounding passenger rail and ferry transit stops and household locations. A half-mile transit stop or household buffer in an urbanized location may include a single census tract, aggregations of multiple census tracts, or, more commonly, portions of multiple census tracts. In order to determine the population and employment characteristics within a buffer using census tract data, a method had to be devised that accommodates the different shapes of the census tracts and the pedestrian buffers.

Solution Method

In order to disaggregate the total population and employment by census tract into the half-mile pedestrian network buffers, we developed a spatial model. Spatial Analyst 9.1 contains tools that can be used to estimate the characteristics of an area using data that represents two different geographies (census tracts and half-mile pedestrian network buffers). The demographic characteristics of census tracts are converted to raster format. The Zonal functions tools take this value raster and the buffers, as inputs and calculate a function or statistic using the value for each raster cell that falls within a zone, represented by the half-mile pedestrian network buffers.[6]

In order to calculate the value raster, we had to first identify a common unit of measurement that could be used to quantify the demographic characteristics

[6] ESRI ArcGIS Desktop Help v. 9.1, *Spatial Analyst Functional Reference--Zonal Based Analysis Tools,* (Redlands: ESRI, 2005).

across the entire geographic area being studied. In this case, we used density (population per unit of area and employment per unit of area) as a common unit of measurement. The density of an area can easily be determined by dividing the total population or employment, by the total land area for that given activity.

We assumed that population and employment are not evenly distributed across the entire geographic extent of a census tract and developed a process that accurately represents the population and employment densities across the study area. We created a correspondence between ABAG's Existing Land Use database and the census tract boundaries. We then created a geometric intersection of the census tract polygon boundaries and the Existing Land Use based polygon boundaries. See Figure 10.

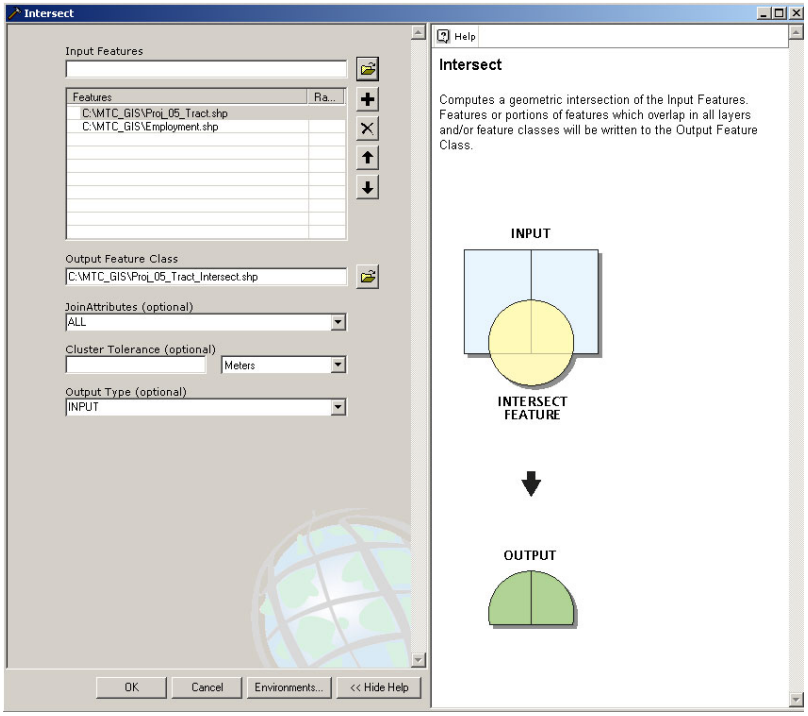


Figure 10. Intersection of Census Tracts and Employment Areas

This process joins the geographic boundaries and their associated attributes into one table, which represents a correspondence between census tract boundaries and existing land use boundaries. We only used the existing land use polygons

associated with residential or employment activities. We assumed that all population patterns occur only on lands identified in the Existing Land Use database for residential activity and that employment patterns occur only on land uses that are identified as commercial or industrial. We then used this information in a spatial model to disaggregate the total population by census tract to residential supporting land uses, and total jobs by census tract to employment supporting land uses. The spatial data used for this analysis had a scale that was set to meters.

To determine the population per square meter, we divided the total population within the defined census tract by the total residential land area (square meters) within that census tract identified as residential land use.

To determine the employment per square meter, we divided the total employment within the defined census tract by the total land area (square meters) within that census tract identified as employment land use.

These datasets were used to create polygonal employment and population feature classes containing attribute values for population per square meter and employment per square meter.

We created a value raster using the Spatial Analyst conversion function-Features to Raster. This tool converts vector-based data into raster-based cells that contain the density values of the polygonal feature classes. The size of each cell is determined by the spatial resolution setting used when converting from feature to raster. Generally speaking, the accuracy of the value raster is dependent on the scale of the data and the size of the cell; the finer the cell resolution and the greater the number of cells that represent small areas, the more accurate the representation.[7] While Spatial Analyst can handle very large raster datasets with millions of cells, we learned that the larger the raster dataset, the longer the

[7] ESRI ArcGIS Desktop Help v. 9.1, Spatial Analyst--Representing Features in a Raster Dataset, (Redlands: ESRI, 2005).

processing time. For the purposes of this analysis we used a cell size of 10 meters. This cell size kept our processing time down to about 2 hours per model run.

Once we created value rasters for employment and population densities, we could easily generate a statistical summary of these values using the Spatial Analyst tool--Zonal Statistics as Table. This tool summarizes the values of a raster within the zones of another dataset and reports the results to a table.[8] The output tables contain summary statistics that include the name of each zone summarized, the total count of all cells within the zone, the total area within the zone, the mean, min and max of all values within the zone, and the total sum of all values within the zone. Using the values for total area (square meters) and mean (average density for the entire zone), we could calculate the total units of population and employment for each zone. This method assumes a constant average density within each of the zones where population- and employment-supporting land uses have been identified.

Average Density = Total Population, Jobs and Households in each tract divided by total square meters of existing residential and employment areas

Many of the region's passenger rail and ferry transit stops are within a half-mile of an adjacent transit station. This results in a transit stop buffer feature class containing several overlapping polygons. The Zonal Statistics as Table tool cannot summarize the statistics for polygons that overlap. When using the tool with overlapping polygons, it will only calculate the statistics for the first buffer it encounters when processing the database. Therefore, we had to develop a process that would summarize the statistics for each station buffer separately- but in a "batch mode".

We used the ModelBuilder tool to develop the basic workflow used to calculate the demographic characteristics for each half-mile pedestrian network buffer. The

[8] ESRI ArcGIS Desktop Help v. 9.1, *Spatial Analyst Functional Reference--Zonal Based Analysis Tools,* (Redlands: ESRI, 2005).

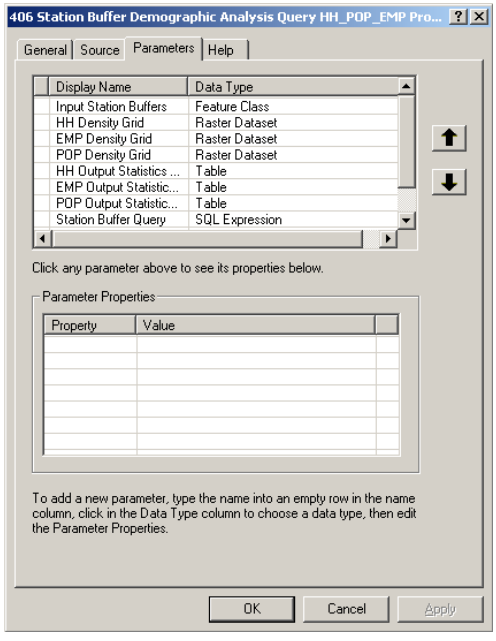
9.1 version of ModelBuilder does not support iterative models, which allow a process to repeat itself using different inputs. Because we needed to summarize each of the overlapping pedestrian buffers separately, we had to export the model into Python and code the iterative looping behavior that would allow us to batch process the entire half-mile pedestrian network buffer dataset. The script, Buffer Area Zonal Statistics, is in Appendix C. Many scripting languages have a common flow control structure, called a "while" loop, that allows the script to loop through successive processes while a statement is true. See Figure 11.

Figure 11. Example of a Python While Loop

```
rows = gp.SearchCursor()
row = rows.Next()
while row:
        # Process that will occur until the rows object has no more instances
row = rows.Next()
```

The script expects nine system arguments: transit stop buffers, a SQL expression used to create a feature layer subset of the buffers, three raster datasets, three output files where the raster tabulations will be stored, and the location of the output file for the buffer selection. These are setup as properties of the script tool. See Figure 12.

Figure 12. Script Tool Properties



As the script runs, first a feature layer is created from input transit stop buffers. The feature layer is copied to an output file for the buffer selection. A temp table is created to hold the second through n-th buffer polygon tabulation prior to appending the results to the output file. A search cursor is opened for the buffer selection output file. If this is the first pass through the search cursor, a feature

layer is made from the first feature in the buffer selection output file, the Zonal Statistics as Table tool is run on each of the three raster datasets, and the outputs are placed in the corresponding output tables. If this is a subsequent pass, the results of the Zonal Statistics as Table tool are appended to the output tables. Last, the counts for each raster are calculated from the Area and Mean fields and added to the output tables. Once the script processes all the buffers in the search cursor, the script terminates.

The Station Buffer Zonal Statistics Script ran within ArcCatalog 9.1. The script ran successfully but each service area took approximately one minute to tabulate. To complete the household records, we set up four PCs and ran the scripts in parallel. This required some manual setup to ensure each PC was processing a different group of records. The output files needed to be appended to a master file at the end of the run.

5. Future Directions for MTC GIS Analyses

Network Analyst 9.2 Enhancements

There are several enhancements scheduled for inclusion in the Network Analyst 9.2 release that will be useful for continued BATS2000 dataset processing. After testing these enhancements, some of the variables described in this paper may be recalculated if more accurate results can be obtained.

We expect to be able to trim Service Area polygons by a specified distance from the street network. This may be helpful in reducing the size of the polygon in areas with few roads, such as mountainous parts of the region and near the shores of the San Francisco Bay. This may produce more accurate categorization of households and work locations into quarter and half mile buffer zones around transit stops.

The ability to produce non-overlapping service areas may be useful for accurately modeling areas with dense transit service, such as San Francisco's MUNI Metro that stops every 3-5 blocks in parts of the city.

Creating network locations using a search tolerance distance will be improved so that the tool first attempts to place the stops at a close distance to network source edges. If no edges are found, then the system will search at increasing distances up to the maximum search tolerance defined. This will improve performance of the location process where many locations are close to a network edge, but some are not, which requires a larger search tolerance. This will be helpful for regional studies such as ours where some point features in rural areas are far from network edges.

Iterative Model Building in ArcGIS 9.2

The framework changes for ModelBuilder in ArcGIS 9.2 will introduce list and series processing, iteration control, and batch processing. The door-to-door trip polyline analysis and the analysis of population and employment densities around transit stops and households required Python scripts to perform iterative looping. While the scripts were fairly simple, because we did not have great familiarity with Python, there were many issues to work out before we could use the scripts to perform the analyses. We look forward to determining if the processes these scripts handled can be replicated in ModelBuilder 9.2, in particular the iteration through a search cursor of unknown record size.

Appendices

Appendix A.
Generate Multiple Routes

```
# ---------------------------------------------------------------------------
# MultiRoute.py
#  This script tool generates routes for a feature class of points.  Each point
#  in the feature class has an associated route identifier field (expected to be an
#  integer field).  The script first determines the unique route id values (using
#  the frequency tool), then loops through the feature class creating a table view
#  containing all the points for a unique route.  A shortest route is then generated
#  for each table view, and this route, and its associated stops, is appended
#  to a new feature class.
#
# Arguments:
#   1 point feature class containing the stops for the route.
#   2 route id feild on (1) containing route id's for each stop.  For example, you
#     may have 100 points and routes 1,2,3,4,5 in the route id field.
#   3 network analysis layer (route layer) containing the settings you want
#     want for the routes, such as find best order, minutes as impedance, etc.
#   4 the output merged paths.  There will be one line feature for each unique route
#   5 the output stops.  This will contain all the information from the analysis (such
#     as cumulative impedance to the stop and the route ID).
#
# This script is provided as a sample only, and mainly as an educational device.
# We anticipate you customizing this for your application.
#
# ---------------------------------------------------------------------------

## typically, we put the entire script in a "try/except" block to catch any and
##  all errors
try:

   # Import system modules
   import sys, string, os, win32com.client, win32api

   # Create the Geoprocessor object
   GP = win32com.client.Dispatch("esriGeoprocessing.GpDispatch.1")

   # We always overwrite outputs for this script
   GP.Overwriteoutput = 1

   # Set the product code - we need ArcInfo to run Frequency
   try:
      GP.SetProduct("ArcInfo")
   except:
      raise Exception, "Could not set product to ArcInfo, which is needed for this tool"

   # Check out any necessary licenses
   GP.CheckOutExtension("Network")

   # Load required toolboxes...
```

```
    #  These statements are not necessary since the geoprocessor loads up all system
toolboxes
    #   automatically on startup, including the toolbox that contains this script.
    #GP.AddToolbox("d:/ArcDev/ArcToolbox/Toolboxes/Analysis Tools.tbx")
    #GP.AddToolbox("d:/ArcDev/ArcToolbox/Toolboxes/Network Analyst Tools.tbx")
    #GP.AddToolbox("d:/ArcDev/ArcToolbox/Toolboxes/Data Management Tools.tbx")

    ## = = = = = = = = = = = = = =
    ## Local routines
    ## = = = = = = = = = = = = = =
    ## -------------
    ## get scratch workspace.  Use the scratch workspace if set to a folder,
    ##   otherwise use the TEMP folder.
    ##
    def getTempWS():
        scratchWS = GP.scratchWorkspace
        if scratchWS:
            desc = GP.Describe(scratchWS)
            if desc.WorkspaceType < > "FileSystem":
                scratchWS = win32api.GetEnvironmentVariable("TEMP")
        else:
            scratchWS = win32api.GetEnvironmentVariable("TEMP")
        return scratchWS

    ## -------------
    ## get temporary scratch table for the frequency tool
    ##  TODO: get a unique name here.
    ##
    def getTempTable():
        scratchWS = getTempWS()
        temp = scratchWS + "\\" + "xxx_MultiRoute.dbf"
        return temp

    ## -------------
    ## get temporary scratch file for holding each route
    ##
    def getTempRoute():
        scratchWS = getTempWS()
        temp = scratchWS + "\\" + "xxx_TempMultiRoute.shp"
        return temp

    ## -------------
    ## get the field object for the input route_id_field from the input
    ##   stops
    ##
    def getField():
        desc = GP.Describe(input_stops)
        fields = desc.Fields

        if not fields:
            msg = "No fields found on %s" % input_stops
```

```
        raise Exception, msg

    field = fields.Next()
    while field:
        name = field.Name
        if name.lower() = = route_id_field.lower():
            return field
        field = fields.Next()

    msg = "Field %s not found on %s" % (route_id_field, input_stops)
    raise Exception, msg

## -------------
## Translate field type into LONG | TEXT | FLOAT | DOUBLE | SHORT | DATE | BLOB
##
def getFieldTypeString(field):
    ftype = field.FieldType
    if not ftype:
        msg = "Null field type"
        raise Exception, msg

    if ftype.lower() = = "smallinteger":
        return "SHORT"
    elif ftype.lower() = = "integer":
        return "LONG"
    elif ftype.lower() = = "single":
        return "FLOAT"
    elif ftype.lower() = = "double":
        return "DOUBLE"

    else:
        msg = "Unknown or illegal field type: %s" % ftype
        raise Exception, msg

    return ""

## = = = = = = = = = = = = = =
## Main module
## = = = = = = = = = = = = = =
# create cursor variables now so in case of an exception, we can test their
#  existance
#
row  = 0
rows = 0

# Get arguments
#  NB: We assume the analysis layer is a route layer, not, say, a Closest Fac layer.
#      This routine will fail if it isn't a route analysis layer.  TODO: use Describe
#      to test layer type.
#
input_stops     = sys.argv[1]   # point fc containing stops
```

```
    route_id_field  = sys.argv[2]   # field containing route id
    analysis_layer  = sys.argv[3]   # route analysis layer
    output_paths    = sys.argv[4]   # ouptut paths line fc
    output_stops    = sys.argv[5]   # output stops point fc

    # Options for AddLocation.  Could easily be made arguments to this function
    #
    field_mappings  = ""    # We will append the route id field mapping
    snap_tolerance  = "100"
    sort_field      = ""
    snap_options    = "SHAPE, MIDDLE, END"
    match_option    = "MATCH_TO_CLOSEST"

    # verbose: if true, we output all messages, a ton of them.  Useful for debugging
    #
    verbose = 1 # 0 = false; 1 = true

    # Get field information now before attempting to use it anywhere.  GetField()
    #  will throw an exception if the field isn't found, and we'll stop execution
    #
    field_info = getField()

    # Get a temporary table for the output of frequency
    #
    temp_table = getTempTable()
    if verbose:
        GP.AddMessage("Temporary frequency table: " + temp_table)

    # Get the frequency of route_id_field.  The output table will contain one
    #  row for each unique route_id.  We'll use this to control looping
    #
    GP.Frequency_analysis(input_stops, temp_table, route_id_field, "")
    if verbose:
        GP.AddMessage(GP.GetMessages())

    # Add the RouteID field to the stops sub-layer.
    #
    field_type = getFieldTypeString(field_info)
    GP.AddFieldToAnalysisLayer_na(analysis_layer, "STOPS", route_id_field, field_type,
field_info.Precision,
                          field_info.Scale, field_info.Length, field_info.Alias,
field_info.IsNullable)
    if verbose:
        GP.AddMessage(GP.GetMessages())

    # We need a temporary scratch route file to copy each RouteID into.  We'll add
    #  a route id field to this temporary file and update this field with the
    #  route id value
    #
    temp_route = getTempRoute()
```

```
   # Construct the name of the stops and routes sub-layers in the analysis layer.  We
   #  need these to copy out the features.  NB: We're using default sublayer names -
   #  If the name of the sublayer has been changed by the user (i.e. "Stops" changed to
   #  "Customers"), this routine will fail.
   #
   route_sublayer = analysis_layer + "\\" + "routes"
   stops_sublayer = analysis_layer + "\\" + "stops"

   # Add the field mapping for our route id for the AddLocations dialog
   #
   field_mappings  = field_mappings + route_id_field + " " + route_id_field + " #"

   # Open a search cursor on this frequency table, loop through each
   #  row creating a table view of just those records with the route id
   #  on the input stops table
   #
   table_view = "temp_table_view"
   rows = GP.SearchCursor(temp_table)

   # fieldval is a variable that will evaluate out to a query.  Example, if route_id_field =
"RunNumber",
   #  then fieldval will be "row.RunNumber".  W/in the loop, "id = eval(fieldval)" will
expand
   #  too "id = row.RunNumber"
   #
   fieldval = "row." + route_id_field

   # ########
   #  Loop for each unique route id
   # ########
   first = 1   # true
   row = rows.Next()
   while row:
      GP.AddMessage("")

      # Get the value of the route id from the input frequency table row and
      #  create an expression for selection
      #
      id = eval(fieldval)
      expression = "[" + route_id_field + "] = %d" % id
      GP.AddMessage(expression)

      # Make a table view of just those records containing the current route id, add these to
      #  the analysis layer and solve.
      #
      GP.MakeTableView(input_stops, table_view, expression , "#", " ")
      if verbose:
         GP.AddMessage(GP.GetMessages())

      GP.AddLocations_na(analysis_layer, "Stops", table_view, field_mappings,
snap_tolerance, sort_field,
```

```
                    snap_options, match_option, "CLEAR")
        if verbose:
           GP.AddMessage(GP.GetMessages())

        GP.Solve(analysis_layer, "SKIP")
        if verbose:
           GP.AddMessage(GP.GetMessages())

        # Write this to the temporary output, add the route id field, and calculate it
        #   For CopyFeatures, the temporary file cannot exist, so delete it
        #
        if GP.Exists(temp_route):
           GP.Delete_management(temp_route, " ")

        GP.CopyFeatures(route_sublayer, temp_route, "", "0", "0", "0")
        if verbose:
           GP.AddMessage(GP.GetMessages())

        GP.AddField_management(temp_route, route_id_field, field_type, field_info.Precision,
field_info.Scale,
                           field_info.Length, field_info.Alias, field_info.IsNullable, "#", "")
        if verbose:
           GP.AddMessage(GP.GetMessages())

        GP.CalculateField(temp_route, route_id_field, `id`)
        if verbose:
           GP.AddMessage(GP.GetMessages())

        # If this is the first pass, copy these temporary features to the output, else
        #  append them to the output
        #
        if first:   # copy
           if GP.Exists(output_paths):
              if verbose:
                 GP.AddMessage("Deleting output %s" % output_paths)
              GP.Delete_management(output_paths, " ")

           GP.CopyFeatures(temp_route, output_paths, "", "0", "0", "0")

           if GP.Exists(output_stops):
              if verbose:
                 GP.AddMessage("Deleting output %s" % output_stops)
              GP.Delete_management(output_stops, " ")

           GP.CopyFeatures(stops_sublayer, output_stops, "", "0", "0")
           if verbose:
              GP.AddMessage(GP.GetMessages())

           first = 0 ## false

        else:   # append
```

```
            GP.Append_management(temp_route, output_paths, "NO_TEST")
            if verbose:
               GP.AddMessage(GP.GetMessages())

            GP.Append_management(stops_sublayer, output_stops, "NO_TEST")
            if verbose:
               GP.AddMessage(GP.GetMessages())

         row = rows.Next()

      # ########
      #  End while loop
      # ########

      # Clean up our temp files
      #
      if GP.Exists(temp_route):
         GP.Delete_management(temp_route, " ")
      if GP.Exists(temp_table):
         GP.Delete_management(temp_table, " ")


## Handle raised exceptions
##
except Exception, errMsg:

      # If we have messages of severity error (2), we assume a GP tool raised it,
      #  so we'll output that.  Otherwise, we assume we raised the error and the
      #  information is in errMsg.
      #
      if GP.GetMessages(2):
         GP.AddError(GP.GetMessages(2))
      else:
         GP.AddError(str(errMsg))
```

Appendix B.
VBA Script to Calculate Multiple Routes

```
Attribute VB_Name = "Module1"
Option Explicit

' data location assumptions:
' - the stops feature class is selected in ArcCatolog
' - at the same level there is a FeatureDataset named FEATURE_DATASET_NAME
' - the feature dataset has a network dataset named NETWORK_DATASET_NAME

Private Const FEATURE_DATASET_NAME As String = "BayNet"
Private Const NETWORK_DATASET_NAME As String = "BayNet10_ND"

' field and network attribute names:

Private Const ROUTEID_FIELD_NAME As String = "tripid"
Private Const TIME_FIELD_NAME As String = "Minutes"
Private Const DISTANCE_FIELD_NAME As String = "Miles"

Private Const TIME_NETATT_NAME As String = "Distance"
Private Const DISTANCE_NETATT_NAME As String = "Drivetime"
Private Const ONEWAY_NETATT_NAME As String = "Oneway"
Private Const HIERARCHY_NETATT_NAME As String = "Hierarchy"   ' not used


Private m_outStatusFieldIndex As Long
Private m_inShapeFieldIndex As Long
Private m_outShapeFieldIndex As Long
Private m_outNameFieldIndex As Long
Private m_inRouteIDFieldIndex As Long
Private m_outRoutesShapeFieldIndex As Long
Private m_inRoutesShapeFieldIndex As Long
Private m_outRoutesTimeFieldIndex As Long
Private m_inRoutesTimeFieldIndex As Long
Private m_outRoutesDistanceFieldIndex As Long
Private m_inRoutesDistanceFieldIndex As Long
Private m_outRoutesTripIDFieldIndex As Long

Sub MultiRoute_BayNet()

  On Error GoTo MsgBoxErrHandler

  ' get inputs - the stops list should be selected

  Dim pGXApplication As IGxApplication
  Set pGXApplication = Application

  Dim pGxObject As IGxObject
  Set pGxObject = pGXApplication.SelectedObject
```

```
  If Not TypeOf pGxObject.InternalObjectName Is IFeatureClassName Then
    MsgBox "Select the stop points before running this tool."
    Exit Sub
  End If

  Dim pName As IName
  Set pName = pGxObject.InternalObjectName

  Dim pInputStopFeatures As IFeatureClass
  Set pInputStopFeatures = pName.Open

  ' Open GDB and NDS

  Dim pDataset As IDataset
  Set pDataset = pInputStopFeatures

  Dim pFWorkspace As IFeatureWorkspace
  Set pFWorkspace = pDataset.Workspace

  Dim pFeatureDataset As IFeatureDataset
  Set pFeatureDataset = pFWorkspace.OpenFeatureDataset(FEATURE_DATASET_NAME)

  Dim pNetworkDataset As INetworkDataset
  Set pNetworkDataset = OpenNetworkDataset(pFeatureDataset,
NETWORK_DATASET_NAME)

  ' Create routes output feature class

  Dim pOutputRoutesFeatureClass As IFeatureClass
  Set pOutputRoutesFeatureClass = CreateRoutesFeatureClass(pFeatureDataset)

  ' Create NAContext and NASolver

  Dim pNAContext As INAContext
  Set pNAContext = CreateSolverContext(pNetworkDataset)

  ' set the snap tolerance (othewrwise the default would be 50m)

  pNAContext.Locator.SnapToleranceUnits = esriMeters

  pNAContext.Locator.SnapTolerance = 100


  ' set up the solver (True for UseOneway, False for UseHierarchy)

  SetSolverSettings pNAContext, True, True

  ' get the output in-memory feature class that we want to populate (Stops)

  Dim pNAStopsClass As INAClass
  Set pNAStopsClass = pNAContext.NAClasses.ItemByName("Stops")
```

```
  ' get the in-memory route results feature class

  Dim pNARoutesClass As INAClass
  Set pNARoutesClass = pNAContext.NAClasses.ItemByName("Routes")

  ' create the output row buffer

  Dim pOutputTable As ITable
  Set pOutputTable = pNAStopsClass

  Dim pOutputCursor As ICursor
  Set pOutputCursor = pOutputTable.Insert(True)

  Dim pOutputRowBuffer As IRowBuffer
  Set pOutputRowBuffer = pOutputTable.CreateRowBuffer

  Dim pRowSubtypes As IRowSubtypes
  Set pRowSubtypes = pOutputRowBuffer

  pRowSubtypes.InitDefaultValues

  ' find the relevant fields

  m_outStatusFieldIndex = pOutputTable.FindField("Status")
  m_outNameFieldIndex = pOutputTable.FindField("Name")
  m_outShapeFieldIndex = pOutputTable.FindField("Shape")
  m_inShapeFieldIndex =
pInputStopFeatures.FindField(pInputStopFeatures.ShapeFieldName)
  m_inRouteIDFieldIndex = pInputStopFeatures.FindField(ROUTEID_FIELD_NAME)

  Dim pRouteResultsFeatureClass As IFeatureClass
  Set pRouteResultsFeatureClass = pNARoutesClass

  m_inRoutesShapeFieldIndex =
pRouteResultsFeatureClass.FindField(pRouteResultsFeatureClass.ShapeFieldName)
  m_inRoutesTimeFieldIndex = pRouteResultsFeatureClass.FindField("Total_" +
TIME_FIELD_NAME)
  m_inRoutesDistanceFieldIndex = pRouteResultsFeatureClass.FindField("Total_" +
DISTANCE_FIELD_NAME)

  m_outRoutesShapeFieldIndex =
pOutputRoutesFeatureClass.FindField(pOutputRoutesFeatureClass.ShapeFieldName)
  m_outRoutesTimeFieldIndex =
pOutputRoutesFeatureClass.FindField(TIME_FIELD_NAME)
  m_outRoutesDistanceFieldIndex =
pOutputRoutesFeatureClass.FindField(DISTANCE_FIELD_NAME)
  m_outRoutesTripIDFieldIndex =
pOutputRoutesFeatureClass.FindField(ROUTEID_FIELD_NAME)

  ' prepare to log errors to the current directory
```

```
Open "SolveErrorLog.txt" For Output As #1

' prepare to write the merged route result features

Dim pRouteFeatureBuffer As IFeatureBuffer
Dim pRouteFeatureCursor As IFeatureCursor

Set pRouteFeatureBuffer = pOutputRoutesFeatureClass.CreateFeatureBuffer
Set pRouteFeatureCursor = pOutputRoutesFeatureClass.Insert(True)

' use a TableSort to sort stops by tripid

Dim pStopsTableSort As ITableSort
Set pStopsTableSort = New esriGeoDatabase.TableSort

pStopsTableSort.Fields = ROUTEID_FIELD_NAME
pStopsTableSort.Ascending(ROUTEID_FIELD_NAME) = True
Set pStopsTableSort.Table = pInputStopFeatures

pStopsTableSort.Sort Nothing

Dim pStopsCursor As ICursor
Set pStopsCursor = pStopsTableSort.Rows

' For each record in the input stops (points)

Dim pStopFeature As IFeature
Dim currRouteID As Long, lastRouteID As Long
Dim count As Long

Dim pGPMessages As IGPMessages
Set pGPMessages = New GPMessages

Dim IsPartialSolution As Boolean    ' existing

Dim IsLastTrip As Boolean          ' new
IsLastTrip = False                 ' new

lastRouteID = -1
count = 0

' add the first stop

Set pStopFeature = pStopsCursor.NextRow

' add the remaining stops
' each time routeID changes, we know to compute the path and clear stops for the next

'if solve fails, proceed to the next trip
On Error GoTo ResumeNextRouteErrHandler
```

```
  While Not pStopFeature Is Nothing

    currRouteID = pStopFeature.Value(m_inRouteIDFieldIndex)

    If (currRouteID < > lastRouteID And lastRouteID < > -1) Then

      count = count + 1

      'solve
      Application.StatusBar.Message(esriStatusMain) = "Solving trip: " + CStr(lastRouteID)
      IsPartialSolution = pNAContext.Solver.Solve(pNAContext, pGPMessages, Nothing)

      'append to results
      AddResultRoute pRouteResultsFeatureClass, pRouteFeatureCursor,
pRouteFeatureBuffer, lastRouteID

      'flush every 100 records
      If (count Mod 100 = 0) Then
        pRouteFeatureCursor.Flush
      End If

NextRouteContinue:

      ' clear stops
      pNAStopsClass.DeleteAllRows

    End If

    AddLocation pStopFeature, pNAContext, pOutputCursor, pOutputRowBuffer

    lastRouteID = currRouteID

    Set pStopFeature = pStopsCursor.NextRow

  Wend

  ' solve the last path             ' existing

  IsLastTrip = True                 ' new

  Application.StatusBar.Message(esriStatusMain) = "Solving trip: " + CStr(lastRouteID)
  IsPartialSolution = pNAContext.Solver.Solve(pNAContext, pGPMessages, Nothing)

  'append last path to results
  AddResultRoute pRouteResultsFeatureClass, pRouteFeatureCursor, pRouteFeatureBuffer,
lastRouteID

  'flush remaining route records
  pRouteFeatureCursor.Flush
```

```
  Exit Sub

MsgBoxErrHandler:

  ' errors will stop execution and display the message
  '
  MsgBox "Error: " & Err.Description
  Exit Sub

ResumeNextRouteErrHandler:

  ' errors are logged to a file
  ' we want all the errors and messages, not just the last like vb Err.Description gives us

  Print #1, "Trip: " & CStr(lastRouteID) & " Error: " & Err.Description & "(" &
CStr(Err.Number) & ")"

  Dim msgCount As Long, i As Long
  Dim pMsg As IGPMessage
  msgCount = pGPMessages.count
  For i = 0 To pGPMessages.count - 1
    Set pMsg = pGPMessages.GetMessage(i)
    Print #1, "      " & pMsg.Description & " (" & CStr(pMsg.ErrorCode) & ")"
  Next

  ' errors will only be reported to the immediate window
  '
  'Debug.Print "Error: " & Err.Description & " Trip: " & CStr(lastRouteID)

  Resume NextRouteContinue

End Sub

Public Function AddLocation(pInputFeature As IFeature, pNAContext As INAContext,
pOutputCursor As ICursor, pOutputRowBuffer As IRowBuffer)

  Dim pLocation As INALocation
  Set pLocation = New NALocation

  Dim Distance As Double
  pNAContext.Locator.QueryLocationByRow pInputFeature, pLocation, Distance

  ' populate the fields in the new stop feature

  Dim pRow As IRow
  Set pRow = pOutputRowBuffer

  If pLocation.IsLocated Then
    pRow.Value(m_outStatusFieldIndex) = esriNAObjectStatusOK
  Else
    pRow.Value(m_outStatusFieldIndex) = esriNAObjectStatusNotLocated
```

```
  End If

  pRow.Value(m_outShapeFieldIndex) = pInputFeature.Value(m_inShapeFieldIndex)

  Dim pNALocationObject As INALocationObject
  Set pNALocationObject = pOutputRowBuffer

  pNALocationObject.NALocation = pLocation

  pOutputCursor.InsertRow pOutputRowBuffer

End Function

'*********************************************************************************
*********
' Open NetworkDataset
'
*********************************************************************************
*******
Public Function OpenNetworkDataset(pFDS As IFeatureDataset, ByVal sNDSName As
String) As INetworkDataset
   Dim pFeatureDatasetExtensionContainer As IFeatureDatasetExtensionContainer
   Dim pFeatureDatasetExtension As IFeatureDatasetExtension
   Dim i As Long
   Dim count As Long
   Dim pDatasetContainer2 As IDatasetContainer2

   ' Get FDS Extension
   Set pFeatureDatasetExtensionContainer = pFDS
   Set pFeatureDatasetExtension =
pFeatureDatasetExtensionContainer.FindExtension(esriDTNetworkDataset)
   Set pDatasetContainer2 = pFeatureDatasetExtension
   Set OpenNetworkDataset =
pDatasetContainer2.DatasetByName(esriDTNetworkDataset, sNDSName)
End Function

'*********************************************************************************
*********
' Create NASolver and NAContext
'*********************************************************************************
*********
Public Function CreateSolverContext(pNetDataset As INetworkDataset) As INAContext
   'Get the Data Element
   Dim pDENDS As IDENetworkDataset
   Dim pDatasetComp As IDatasetComponent

   Set pDatasetComp = pNetDataset
   Set pDENDS = pDatasetComp.DataElement

   Dim pNASolver As INASolver
   Dim pContextEdit As INAContextEdit
```

```
    Set pNASolver = New esriNetworkAnalyst.NARouteSolver
    Set pContextEdit = pNASolver.CreateContext(pDENDS, "Route")
    pContextEdit.Bind pNetDataset, New GPMessages

    Set CreateSolverContext = pContextEdit
End Function

'*********************************************************************************
*********
' Set Route Solver Settings
'*********************************************************************************
*********
Public Sub SetSolverSettings(ByRef pContext As INAContext, _
                             ByVal bOneWay As Boolean, _
                             ByVal bUseHierarchy As Boolean)

    'Set Route specific Settings
    Dim pSolver As INASolver
    Set pSolver = pContext.Solver

    Dim pRteSolver As INARouteSolver
    Set pRteSolver = pSolver

    pRteSolver.OutputLines = esriNAOutputLineTrueShapeWithMeasure
    pRteSolver.CreateTraversalResult = True
    pRteSolver.UseTimeWindows = False
    pRteSolver.FindBestSequence = False
    pRteSolver.PreserveFirstStop = False
    pRteSolver.PreserveLastStop = False


    'Set generic Solver settings
    ' set the impedance attribute
    Dim pSolverSettings As INASolverSettings
    Set pSolverSettings = pSolver
    pSolverSettings.ImpedanceAttributeName = TIME_NETATT_NAME

    Dim pAccumulateNames As IStringArray
    Set pAccumulateNames = New StrArray
    pAccumulateNames.Add DISTANCE_NETATT_NAME

    Set pSolverSettings.AccumulateAttributeNames = pAccumulateNames

    ' Set the OneWay Restriction if necessary
    Dim restrictions As IStringArray
    Set restrictions = pSolverSettings.RestrictionAttributeNames
    restrictions.RemoveAll
    If bOneWay Then
       restrictions.Add ONEWAY_NETATT_NAME
    End If
    Set pSolverSettings.RestrictionAttributeNames = restrictions
```

```
    'Restrict UTurns
    pSolverSettings.RestrictUTurns = esriNFSBNoBacktrack

    ' Set the Hierachy attribute
    pSolverSettings.UseHierarchy = bUseHierarchy
    If bUseHierarchy Then
       pSolverSettings.HierarchyAttributeName = HIERARCHY_NETATT_NAME
       pSolverSettings.HierarchyLevelCount = 3
       pSolverSettings.MaxValueForHierarchy(1) = 1
       pSolverSettings.NumTransitionToHierarchy(1) = 9

       pSolverSettings.MaxValueForHierarchy(2) = 2
       pSolverSettings.NumTransitionToHierarchy(2) = 9
    End If

    ' Do not forget to update the context after you set your impedance
    Dim pDatasetComp As IDatasetComponent
    Set pDatasetComp = pContext.NetworkDataset

    pSolver.UpdateContext pContext, pDatasetComp.DataElement, New GPMessages

    ' Update the StreetDirectionAgent context
    Dim pNAAgent As INAAgent
    Set pNAAgent = pContext.Agents.ItemByName("StreetDirectionsAgent")
    pNAAgent.OnContextUpdated
End Sub

Public Function CreateRoutesFeatureClass(pFDS As IFeatureDataset) As IFeatureClass

  Dim pFields As IFields
  Dim pFieldsEdit As IFieldsEdit
  Dim pGeomDef As IGeometryDef
  Dim pGeomDefEdit As IGeometryDefEdit
  Dim pField As IField
  Dim pFieldEdit As IFieldEdit
  Dim strShapeFld As String
  Dim j As Integer

  On Error GoTo EH

  Set CreateRoutesFeatureClass = Nothing
  If pFDS Is Nothing Then Exit Function

  Dim pCLSID As UID
  Set pCLSID = New UID
  pCLSID.Value = "esriGeoDatabase.Feature"

  Set pFieldsEdit = New Fields

  '' create the geometry field
```

```
Set pGeomDef = New GeometryDef
Set pGeomDefEdit = pGeomDef

'' assign the geometry definiton properties.
With pGeomDefEdit
  .GeometryType = esriGeometryPolyline
  .GridCount = 1
  .GridSize(0) = 10
  .AvgNumPoints = 2
  .HasM = False
  .HasZ = False
End With

Set pField = New Field
Set pFieldEdit = pField

pFieldEdit.Name = "shape"
pFieldEdit.AliasName = "geometry"
pFieldEdit.Type = esriFieldTypeGeometry
Set pFieldEdit.GeometryDef = pGeomDef
pFieldsEdit.AddField pField

'' create the object id field
Set pField = New Field
Set pFieldEdit = pField
pFieldEdit.Name = "OBJECTID"
pFieldEdit.AliasName = "object identifier"
pFieldEdit.Type = esriFieldTypeOID
pFieldsEdit.AddField pField

'' create the total time
Set pField = New Field
Set pFieldEdit = pField
pFieldEdit.Name = TIME_FIELD_NAME
pFieldEdit.Type = esriFieldTypeDouble
pFieldsEdit.AddField pField

'' create the total distance
Set pField = New Field
Set pFieldEdit = pField
pFieldEdit.Name = DISTANCE_FIELD_NAME
pFieldEdit.Type = esriFieldTypeDouble
pFieldsEdit.AddField pField

'' create the TripID
Set pField = New Field
Set pFieldEdit = pField
pFieldEdit.Name = ROUTEID_FIELD_NAME
pFieldEdit.Type = esriFieldTypeInteger
pFieldsEdit.AddField pField
```

```
  Set pFields = pFieldsEdit

  ' locate the shape field
  For j = 0 To pFields.FieldCount - 1
    If pFields.Field(j).Type = esriFieldTypeGeometry Then
      strShapeFld = pFields.Field(j).Name
    End If
  Next

  Dim fcName As String
  fcName = "RouteResults" + CStr(Format(Now, "yyyymmddhhmmss"))

  Set CreateRoutesFeatureClass = pFDS.CreateFeatureClass(fcName, pFields, pCLSID,
Nothing, esriFTSimple, strShapeFld, "")

  Exit Function
EH:
    MsgBox Err.Description, vbInformation, "createDatasetFeatureClass"
End Function

Public Function AddResultRoute(pNARouteFeatures As IFeatureClass, pFeatureCursor As
IFeatureCursor, pFeatureBuffer As IFeatureBuffer, tripID As Long)

  ' get the one and only route result record

  Dim pCursor As IFeatureCursor
  Dim pFeature As IFeature

  Set pCursor = pNARouteFeatures.Search(Nothing, False)
  Set pFeature = pCursor.NextFeature

  ' set values

  Dim pRow As IRow
  Set pRow = pFeatureBuffer

  pRow.Value(m_outRoutesShapeFieldIndex) =
pFeature.Value(m_inRoutesShapeFieldIndex)
  pRow.Value(m_outRoutesTimeFieldIndex) = pFeature.Value(m_inRoutesTimeFieldIndex)
  pRow.Value(m_outRoutesDistanceFieldIndex) =
pFeature.Value(m_inRoutesDistanceFieldIndex)
  pRow.Value(m_outRoutesTripIDFieldIndex) = tripID

  ' insert the feature

  pFeatureCursor.InsertFeature pFeatureBuffer

End Function
```

Appendix C.
Station Buffer Zonal Statistics

```
## = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
## Station Buffer Zonal Statistics Query HH-EMP-POP.py
## Created on: Mon Oct 10 2005 11:13:59 AM
## = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

## Import system modules
import sys, string, os, win32com.client

## Create the Geoprocessor object
gp = win32com.client.Dispatch("esriGeoprocessing.gpDispatch.1")

## Check out necessary licenses for tools that are used in script
gp.CheckOutExtension("Spatial")

## Load required toolboxes...This step is not necessary as the geoprocessor object adds
##all necessary toolboxes
##gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Spatial Analyst Tools.tbx")
##gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Network Analyst Tools.tbx")
##gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Data Management Tools.tbx")

## = = = = = = = = = = = = =
## Local routines
## = = = = = = = = = = = = =
## -------------
## Get scratch workspace.  Use the scratch workspace if set to a folder,
## otherwise use the TEMP folder.
##
def getTempWS():
   scratchWS = gp.scratchWorkspace
   if scratchWS:
      desc = gp.Describe(scratchWS)
      if desc.WorkspaceType < > "FileSystem":
         scratchWS = "C:\\Network_Temp"
   else:
      scratchWS = "C:\\Network_Temp"
   return scratchWS

## -------------
## Get temporary scratch table for holding results of Tabulate Data function
## for each iteration.
##
def get_TempTable():
   scratchWS = getTempWS()
   temp = scratchWS + "\\" + "xxx_Tabulat_polygon.dbf"
   gp.AddMessage(gp.GetMessages())
   return temp

## System Arguments
input_buffers = sys.argv[1]
HH_Density_Grid = sys.argv[2]
EMP_Density_Grid = sys.argv[3]
POP_Density_Grid = sys.argv[4]
```

```
HH_output_dbf = sys.argv[5]
EMP_output_dbf = sys.argv[6]
POP_output_dbf = sys.argv[7]
Station_Buffer_Expression = sys.argv[8]
## Output Station Buffers
Select_Station_Buffers_shp = sys.argv[9]
##  Local variables...
facility_id_field = "FacilityID"
input_buffer_layers = "Input_Locations_Polygon"
input_locations_layer = "Input Locations"
input_buffers_selection = "Input Buffers"
Select_Station_Buffers = "Input Buffers Selection"

## = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
## Select Station Buffers for Analysis
## = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

##Process: Make Feature Layer...Select Buffers for Analysis
gp.MakeFeatureLayer_management(input_buffers, Select_Station_Buffers,
Station_Buffer_Expression, "", "ObjectID ObjectID VISIBLE;FacilityID FacilityID VISIBLE;Name Name
VISIBLE;FromBreak FromBreak VISIBLE;ToBreak ToBreak VISIBLE;pop2K pop2K VISIBLE;hh2K hh2K
VISIBLE;job2K job2K VISIBLE;pop_d pop_d VISIBLE;hh_d hh_d VISIBLE;job_d job_d VISIBLE;emp_a
emp_a VISIBLE;res_a res_a VISIBLE")
gp.AddMessage(gp.GetMessages())

##Process: Copy Features...
gp.CopyFeatures_management(Select_Station_Buffers, Select_Station_Buffers_shp, "", "0", "0", "0")
gp.AddMessage(gp.GetMessages())

## = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
## Create temp table which will hold the 2nd through nth polygon tabulation prior
## to appending record to Tabulat_polygon_dbf
## = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
temp_table = get_TempTable()
gp.AddMessage("Temporary Table: " + temp_table)

#### = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
#### Create Expression that is used to select one polygon from the polygon feature class
#### Process will be finished within loop
####
#### fieldval is a variable that will evaluate out to a query.  Example, if FacilityID_field =
"RunNumber",
#### then fieldval will be "row.RunNumber".  W/in the loop, "id = eval(fieldval)" will expand
#### to "id = row.RunNumber"
#### = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
fieldval = "row." + facility_id_field
##
#### = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
#### Create search cursor on input_locations feature class to control looping
#### = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

rows = gp.SearchCursor(Select_Station_Buffers_shp)
gp.AddMessage("Search Cursor Created")

## = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
## Start loop
```

```
## = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

first = 1 # True
count = 0
row = rows.Next()
while row:
   count = count + 1
   gp.AddMessage(count)

   ## Get the value of the facility id field from the current row of the
   ## polygon feature class and create an expression for feature layer selection
   ##
   ## expression = "" + facility_id_field + " = %d" % id
   ## is used for creating a SQL expression on a shapefile
   ##
   ## expression = "[" + facility_id_field + "] = %d" % id
   ## is used for creating a SQL expression on a personal geodatabase file

   id = eval(fieldval)
   expression = "" + facility_id_field + " = %d" % id
   gp.AddMessage(expression)

   #rowvalue = row.GetValue(facility_id_field)
   #gp.AddMessage(rowvalue)

   gp.MakeFeatureLayer_management (Select_Station_Buffers, input_buffer_layers, expression, "",
"")
   gp.AddMessage(gp.GetMessages())

   # Testing Process Only: To check if expression was built correctly and a feature layer was
returned
   # gp.CopyFeatures_management (input_buffer_layers, input_buffer_layers_Check, "", "0", "0",
"0")
   # gp.AddMessage(gp.GetMessages())


   ## If this is the first pass, write the tabulate results to output table, else
   ## write the results to a temp table and then append row to the output table

   if first:   ## write results to output table

      ## Process: Zonal Statistics as Table...
      gp.ZonalStatisticsAsTable_sa(input_buffer_layers, "Name", HH_Density_Grid, HH_output_dbf,
"DATA")
      gp.AddMessage(gp.GetMessages())
      gp.ZonalStatisticsAsTable_sa(input_buffer_layers, "Name", EMP_Density_Grid,
EMP_output_dbf, "DATA")
      gp.AddMessage(gp.GetMessages())
      gp.ZonalStatisticsAsTable_sa(input_buffer_layers, "Name", POP_Density_Grid,
POP_output_dbf, "DATA")
      gp.AddMessage(gp.GetMessages())
      if gp.Exists(input_buffer_layers):
         gp.Delete_management(input_buffer_layers, " ")
      first = 0 ## false

   else:   ## append results to output table3
```

```
        ## Calculate Household Statistics
        gp.ZonalStatisticsAsTable_sa(input_buffer_layers, "Name", HH_Density_Grid, HH_temp_table,
"DATA")
        gp.AddMessage(gp.GetMessages())
        gp.Append_management(HH_temp_table, HH_output_dbf, "TEST")
        gp.AddMessage(gp.GetMessages())
        if gp.Exists(HH_temp_table):
            gp.Delete_management(HH_temp_table, " ")
        ## Calculate Employment Statistics
        gp.ZonalStatisticsAsTable_sa(input_buffer_layers, "Name", EMP_Density_Grid,
EMP_temp_table, "DATA")
        gp.AddMessage(gp.GetMessages())
        gp.Append_management(EMP_temp_table, EMP_output_dbf, "TEST")
        gp.AddMessage(gp.GetMessages())
        if gp.Exists(EMP_temp_table):
            gp.Delete_management(EMP_temp_table, " ")
        ## Calculate Population Statistics
        gp.ZonalStatisticsAsTable_sa(input_buffer_layers, "Name", POP_Density_Grid,
POP_temp_table, "DATA")
        gp.AddMessage(gp.GetMessages())
        gp.Append_management(POP_temp_table, POP_output_dbf, "TEST")
        gp.AddMessage(gp.GetMessages())
        if gp.Exists(POP_temp_table):
            gp.Delete_management(POP_temp_table, " ")
        ## Delete temporary input buffer file before next loop
        if gp.Exists(input_buffer_layers):
            gp.Delete_management(input_buffer_layers, " ")

    row = rows.Next()

##= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
## Add field to calculate totals
##= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

# Process: Add Field for Totals...
gp.AddField_management(HH_output_dbf, "HH2K", "LONG", "", "", "", "", "NON_NULLABLE",
"NON_REQUIRED", "")
gp.AddMessage(gp.GetMessages())
gp.AddField_management(EMP_output_dbf, "JOB2K", "LONG", "", "", "", "", "NON_NULLABLE",
"NON_REQUIRED", "")
gp.AddMessage(gp.GetMessages())
gp.AddField_management(POP_output_dbf, "POP2K", "LONG", "", "", "", "", "NON_NULLABLE",
"NON_REQUIRED", "")
gp.AddMessage(gp.GetMessages())
# Process: Add Field for EMP/RES Acres...
gp.AddField_management(HH_output_dbf, "RES_A", "DOUBLE", "", "", "", "", "NON_NULLABLE",
"NON_REQUIRED", "")
gp.AddMessage(gp.GetMessages())
gp.AddField_management(POP_output_dbf, "RES_A", "DOUBLE", "", "", "", "", "NON_NULLABLE",
"NON_REQUIRED", "")
gp.AddMessage(gp.GetMessages())
gp.AddField_management(EMP_output_dbf, "CI_ACRES", "DOUBLE", "", "", "", "",
"NON_NULLABLE", "NON_REQUIRED", "")
gp.AddMessage(gp.GetMessages())
# Process: Calculate HH/EMP/POP Totals...
gp.CalculateField_management(HH_output_dbf, "HH2K", "[Area]*[Mean]")
```

```
gp.AddMessage(gp.GetMessages())
gp.CalculateField_management(EMP_output_dbf, "JOB2K", "[Area]*[Mean]")
gp.AddMessage(gp.GetMessages())
gp.CalculateField_management(POP_output_dbf, "POP2K", "[Area]*[Mean]")
gp.AddMessage(gp.GetMessages())
# Process: Calculate HH/EMP/ Net Acre Totals...
gp.CalculateField_management(HH_output_dbf, "RES_A", "[Area]*0.000247")
gp.AddMessage(gp.GetMessages())
gp.CalculateField_management(EMP_output_dbf, "RES_A", "[Area]*0.000247")
gp.AddMessage(gp.GetMessages())
gp.CalculateField_management(POP_output_dbf, "CI_ACRES", "[Area]*0.000247")
gp.AddMessage(gp.GetMessages())
```